ToiletPaper #137

API-Calls mit React Query

Autor: Benjamin Hogl / Software Engineer / Standort Stuttgart

✗ Problem

Dein React-Frontend muss mit einem Backend reden? Du willst nicht extra einen Redux-Store mit unzähligen Actions und Reducern einrichten, nur um die Antworten des Backends und den Status der Requests irgendwo zu speichern? Du willst dich nicht darum kümmern müssen, wann welcher Request gestartet werden muss, sondern einfach auf die Daten zugreifen können? Du willst keine komplizierte Logik entwickeln, um die angezeigten Daten aktuell zu halten oder fehlgeschlagene Requests zu wiederholen?

✓ Lösung

React Query kümmert sich um fast alles, was mit HTTP-Queries zu tun hat, darunter z. B.:

- Caching und Request-Deduplizierung über eindeutige queryKeys (im Beispiel unten `["users", 3]`)
- Automatische Hintergrund-Aktualisierung der Daten (konfigurierbar)
- Automatische Wiederholung, wenn ein Request fehlschlägt (ebenfalls konfigurierbar)
- Bereitstellung des aktuellen Request-Status (`isLoading`, `isError`, `isSuccess` usw.)

Wenn ein Request nicht sofort beim Mounten einer Komponente gestartet werden soll, kann er als deaktiviert markiert werden.

Das Rendern von Inhalten, Loading Indicators oder Fehlermeldungen bleibt dir überlassen, ebenso wie die Implementierung des eigentlichen HTTP-Requests (z. B. mit axios oder über die Fetch API). In diesen Punkten macht React Query dir keine Vorgaben.

Durch den Einsatz von React Query wird dein Code höchstwahrscheinlich übersichtlicher und weniger fehleranfällig. Und für den Fall, dass doch mal etwas nicht so funktioniert wie gewünscht, bringt React Query sogar eigene Devtools mit. Die werden bei einem Production-Build automatisch entfernt, darum musst du dich also nicht kümmern.

➔ Beispiel

```tsx
import { QueryClient, QueryClientProvider, useQuery } from "react-query";
import { ReactQueryDevtools } from "react-query/devtools";
type ApiResponse = { data: { first_name: string } };

const queryClient = new QueryClient();
export default function App() {
  return (
    <QueryClientProvider client={queryClient}>
      <Example />
      <ReactQueryDevtools />
    </QueryClientProvider>
  );
}
function Example() {
  const query = useQuery<ApiResponse>(["users", 3], () => {
    return fetch("https://reqres.in/api/users/3?delay=1").then((res) => res.json());
  });
  if (query.isLoading) return <p>Loading...</p>;
  if (query.isError) return <button onClick={() => query.refetch()}>Try again</button>;
  return <p>Hello {query.data?.data.first_name}</p>;
}
```

+ Weiterführende Aspekte

- https://react-query.tanstack.com/guides/important-defaults
  Die Default-Einstellungen wurden sorgfältig gewählt, können aber verwirrend sein, wenn man nichts davon weiß.
- https://epicreact.dev/my-state-management-mistake von Kent C. Dodds
  Relevantes Zitat daraus: *"Server cache is not the same as UI state, and should be handled differently."*